: Assume the following three definitions ...
(define list.A (list 1 (list (list 2 3) 4 5) (list) (list 6 (list 7 8))))
(define list.B (list 1 (list 1 (list) 2 (list 3)) 4 (list (list 5 6 7) 8)))
(define list.C (list (list (list 1) (list 2 3)) (list) (list 4 5) (list (list 6 7)) 8))

: Here's the literal for listA again, but with a box drawn around each element: (list 1 (list (list 2 3) 4 5) (list) (list 6 (list 7 8)))
: Write out the literals for listB and listC again, with a box drawn around each element ...
Beside each of the following expressions show its value ...

(reverse list.A)

(reverse list.B)

(reverse list.C)

; Show, with standard underlining, the steps for:  
(step (map reverse (rest (list 1 (list 2 3) 4 5) (list) (list 6 (list 7 8)))))

; Assume the following definition has been entered:run ...

(define (maybe-length v)
  (if (list? v) (length v)
       (-))) ; that "-" is an image, from (solid-rectangle 8 1)

; ... then show the steps, with standard underlining, for ...
(step (maybe-length "coffee"))

(step (maybe-length (list "coffee" "tea" "water")))

(step (hide maybe-length) (map maybe-length (list "coffee" "tea" "water")))
(step (hide maybe-length) (map maybe-length (list (list 1 2 3 4 5) (list) (list 6 7))))

(step (hide maybe-length) (map maybe-length (list (list (list 1 2) (list 3)) 4 (list (list 5 6 7) 8))))

(step (hide maybe-length) (map maybe-length (list (list (list 1) (list 2 3) (list)) (list 4 5) (list (list 6 7) 8))))